# DATA MINING (CSE572) ASSIGNMENT #2: USER DEPENDENT AND USER INDEPENDENT ANALYSIS

Cristina Maldonado

ARIZONA STATE UNIVERSITY (1216168421)

February 27, 2019

**Contents**

**Introduction**

This assignment was created to bring awareness to differences between user dependent analysis and user independent analysis of EMG signals using three different machine learning algorithms for classification. The data consists of 8 EMG signals' sensor information and ground truth data that indicate when the user is eating. This is a binary classification problem, the EMG signals capture eating or non-eating data. The three machine learning algorithms that are used to predict the class labels are decision trees, support vector machines, and neural networks.

**Background Information: Pre-phase code optimization**

The code imported from the previous assignment, which focused on feature extraction and principle component analysis, was used as the base for this assignment. Python 3.7.3 and Jupyter Notebooks was utilized.

Code was optimized by performing three strategies, 1) limiting the use of for loops, 2) reducing memory footprint, and 3) using local variables.

The use of for loops was replace with similar but faster "loops" like maps, list comprehensions, and built in iteration for data structures. All calculations were moved outside of the for loops where possible. This was accomplished by declaring variables beforehand, calling functions outside of the loop, or doing calculations after the loop.

The second step reducing the memory footprint. Unnecessary copy of dataframe was removed, copies which increased performed time. Data structures that could be combined were combined, for example, similarly calculated statistical features. String manipulations that were time and memory intensive like "string + string1" were modified by creating a new string from a list of strings.

Finally, the use of global variables was eliminated and instead variables declared and called locally in the function. Although not in tune with keeping a low memory footprint, local variable usage increased run time. Additionally, it decreased the likelihood of inadvertent changes in the variable that could lead to erroneous calculations/information.

These are the pre-phase 1 and 2 code optimization routes that were implemented. Further optimization code routes will be discussed in Phase 1 or Phase 2.

**Phase 1: User dependent analysis**

The purpose of this phase is to train a) decision tree, support vector machine, and neural network with training data and use test data to report accuracy metrics of precision, recall, and f1 score.

The training and testing data sets consist applying the PCA output to a feature set. The feature set is divided in two parts: a) train data set(from each user in the data set: 60% of combined activities; equal parts eating and non-eating ), and b):test data set(from each user in the data set: 40% of combined activities; equal parts eating and non-eating ).

**Phase 1.1: Gathering the data**

All the data for both phases of the assignment was captured in one loop. Different from the previous assignment, a more robust method to catch data that is not formatted per usual is implemented. The new method accounts for the wrong number of columns in a line, multiple delimiters (eg. sep=r'\,|\t' ), and moving past users not within specified range. The data for each user is preprocessed and mapped resulting in a eat_df and noeat_df. The split function is called

for each user's eat_df and noeat_df. The function returns two dataframes, train and test which is appended to a list at the end of each for loop iteration.

The following is an example where code is optimized significantly. In the previous assignment the user dataframe was appended to the master_df:

```
for i in range(9, 42):
        ground_f, ground_s, emg_f, emg_s = load_data(user)
        eat_df, noeat_df, all_df =
combine_activity(ground_f, ground_s, emg_f, emg_s)
        master_df = pd.concat([master_df, all_df], axis=0
```

There was a performance bottleneck area in the concatenation of the dataframe to the master dataframe. Appending to a dataframe is achieved through quadratic copy, it returns a copy of the original dataframe plus the new row. [1] The complexity of that operation is $O(n^2)$, more time consuming the larger the number of samples in a dataframe. The bottleneck was address with:

```
for i in range(9, 42):
        ground_f, ground_s, emg_f, emg_s = load_data(i)
        comb, eat_df, noeat_df = combine_activity( … )
        train, test = split_train_test(eat_df, noeat_df)
        train_list.append(train)
```

Appending to a list has a complexity of O(n) and only invoked pd.concat() once outside of the for loop. [5]

**Phase 1.2: Dividing 60/40 split**

In order to create a train and test dataset the split_train_test(e, n, split=0.60) function is called. The function is called with parameters eat_df and noeat_df for each user. The function can account for different percentage of splits, but defaults to 0.60. The dataframes are split on the index that is calculated by split_row = int(len(e.index) * split ). The result of doing that on the two dataframes is 60% of eat_df, 60% of noeat_df, 40% of eat_df and 40% of noeat_df. The 60% dataframes will be combined as well as the remaining two. The two dataframe for each user then becomes a part of the train and test.

**Phase 1.3: Applying the feature extraction**

In the user_dep(test_list, train_list) function the dataframe for both train and test is created. The dataframe are then sent to get_pca_df(), where it sends a portion of the dataframe to the function that will prepare the dataframe with new features. The features that are used are minimum, maximum, standard deviation, root mean square, and average rectified value. These are the same values that were used for assignment 1. For the purpose of this assignment slight modifications were made.

One such modification is calculating minimum, maximum, and standard deviation in one function. In the previous assignment, each dataframe was iterated through each column in a for loop. For each iteration of the for loop statistical measures were calculated using the describe() function. Finally, minimum, maximum, and standard deviation values were assigned to three

dataframe respectively. For this assignment, all the data is gathered using column-wise operations. The describe() function is called once, and the corresponding columns for min, max, and std, are appended to one dataframe with 24 columns (8 EMG sensors * 3 statistical measures).

Also different from the first assignment is the slight modification of the average rectified value function, get_arv(). Formerly a filtering function from the SciPy library, filtfilt, which applies a two-time digital filter going forward and backward, was used. [4] However during this assignment there was a padlen error that would be thrown on certain sections of the dataframe. In lieu of the filtfilt() function, another filtering function from the same library is used, lfilter() function.

All other functions of extracting a feature dataframe remain the same as the previous assignment. The feature dataframe that results is used in the PCA function. About 95% of the variance is captured within 2 components, thus 2 component PCA dataframe is chosen and returned. The PCA dataframe is finally ready to be used in the machine learning algorithms.

**Phase 1.4: Decision Tree**

Skikit-Learn's tree library for the DecisionTreeClassifier is used for this assignment. The training data (X_train and y_train) is trained/fitted on the tree classifier. The predict method of the DecisionTreeClassifier class is used to make predictions on the test data (X_test). Evaluation of the of the algorithm is carried out using Skikit-Learn's handy metrics library. The following figures are pulled: precision, recall, and f1.

**Phase 1.5: Support Vector Machine**

Skikit-Learn's LinearSVC class is used for the support vector machine algorithm. There exists a SVC class that can implement a linear SVC by specifying it in the parameters. Attempts were made to use SVC with a linear kernel; however, the model took an extremely long time to converge 2+ hours. Additionally, attempts were made to tweak the data going in by using larger chucks of data to calculate the feature (therefore smaller n samples) and increasing the cache. LinearSVC was found to be a suitable replacement for SVC, because it could scale better for large n number of samples and it could handle two classes (eating and non-eating). The classifier is trained/fitted on the same data that is used for the other learning algorithms (X_train and y_train), and evaluation is calculated as before.

**Phase 1.6: Neural Network**

Skikit-Learn's multi-layer perceptron classifier, MLPClassifier was used for the neural network portion of the assignment. Again, the same train and test data is utilized in this learning algorithm. With the MLPClassifier, the number of hidden layers can be specified in the hidden_layer_sizes parameter. After experimenting with different hidden layer size tuple, the tupple size for this classifier is hidden_layer_sizes=(10, 10, 10). Three layers of 10 nodes each. The best activation function for binary classification per the implementation used in Sikit-Learn is 'logistic' and that is what is specified in this assignment. 'logistics' ensures that the output values are either 1 or 0. Finally, the max iterations are set to 1000; the default is 200, however the data would not converge default iteration of 200. The metrics are evaluated as before.

**Phase 2: User dependent analysis**

The purpose of this phase is the same as before to train machine learning algorithms decision tree, support vector machine, and neural network with training data and use test data to test the accuracy of those machine learning algorithms. I will finally report the accuracy metrics of precision, recall, and f1 score.

The key difference is that the training and testing data set is divided 60%/40% from a pool of *all* the users' combined activities feature matrix. As before I will a): train 60% of the training data and b): test the remaining 40% of the data.

**Phase 2.1: Dividing 60/40 split**

For this assignment I utilized the Scikit-Learn library's model_selection library to split the data into the train and test data set. X is the data portion of the feature dataframe and y is the classification for each row in the feature dataframe.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.40)
```

**Phase 2.2: Feature extraction, PCA, and 3 MLA.**

The methods for feature extraction, PCA, and the machine learning algorithms are identical to the ones found in Phase 1.

**Results**

Weighted measurement accounts for class imbalance by computing the average of binary metrics in which each class's score is weighted by its presence in the true data sample.

Micro gives each sample-class pair an equal contribution to the overall metric.Micro-averaging may be preferred in multilabel settings, including multiclass classification where a majority class is to be ignored.

| Weighted | | Precision | Recall | F1 |
|---|---|---|---|---|
| **User Dependent Analysis** | SVM | 0.14 | 1.0 | 0.25 |
| | Tree | 0.77 | 0.53 | 0.60 |
| | NN | 0.78 | 0.54 | 0.61 |
| **User Independent Analysis** | SVM | 0.75 | 1.0 | 0.86 |
| | Tree | 0.67 | 0.67 | 0.67 |
| | NN | 0.79 | 0.77 | 0.70 |

| Micro | | Precision | Recall | F1 |
|---|---|---|---|---|
| **User Dependent Analysis** | SVM | 0.14 | 1.0 | 0.25 |
| | Tree | 0.54 | 0.54 | 0.54 |
| | NN | 0.60 | 0.60 | 0.60 |
| **User Independent Analysis** | SVM | 0.74 | 1.0 | 0.85 |
| | Tree | 0.68 | 0.68 | 0.68 |
| | NN | 0.77 | 0.77 | 0.77 |

**Conclusion**

User dependent analysis using SVM is the worst measurement in almost all cases. It has a high recall, but low precision. This indicates that most of the predicted results returned were labeled incorrectly. User dependent analysis using NN was the best of the three MLA's used.

User independent all came back with pretty similar results. The best results are with SVM and NN. However, NN has faster performance it might be chosen over SVM (especially with multi-class datasets).

Compared to the two, user independent training/testing yields more accurate results.

**References**

[1] "*API Reference",* The Panda Community, Aug. 2018. Accessed on: Feb 20, 2020. [Online]. Available: https://pandas.pydata.org/pandas-docs/version/0.23.4/api.html

[2] Bob, *5 Tips to Speed Up Your Python Code,* Feb. 2017. Accessed on: Feb. 21, 2020. [Online]. Available: https://pybit.es/faster-python.html

[3] F. Pedregosa et al., *Scikit-learn: Machine Learning in Python,* Oct. 2011. Accessed on: Feb. 21, 2020. [Online]. Available: https://scikitlearn.org/stable/modules/neural_networks_supervised.html

[4] "*Scipy.signal.filtfilt*," The SciPy Community, Dec. 2019. Accessed on: Feb. 20, 2020. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html #scipy.signal.filtfilt

[5] "*Time Complexity,"* Python™, Jun. 2016. Accessed on: Feb. 24, 2020. [Online] Available: https://wiki.python.org/moin/TimeComplexity